Maximum subarray problem

Given an array, find a non-empty contiguous subarray that maximizes its sum:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>-3</td>
<td>-25</td>
<td>20</td>
<td>-3</td>
<td>-16</td>
<td>-23</td>
<td>18</td>
<td>20</td>
<td>-7</td>
<td>12</td>
<td>-5</td>
<td>-22</td>
<td>15</td>
<td>-4</td>
<td>7</td>
</tr>
</tbody>
</table>

maximum subarray
Maximum subarray problem

$$4 \quad -6 \quad 7 \quad 2 \quad 5 \quad -1 \quad 3 \quad 2$$

left mid right
Maximum subarray problem

4 -6 7 2 5 -1 3 2

4 -6 7 2  mid  right
Maximum subarray problem

```plaintext
4 -6 7 2 5 -1 3 2
```

```
mid
```

```
right
```

```
4 -6 7 2
```

```
mid
```

```
7 2
```

```
4 -6
```
Maximum subarray problem
Maximum subarray problem
Maximum subarray problem

4 -6 7 2 5 -1 3 2

4 -6 7 2 mid right

4 mid 7 2

7 mid 2
Maximum subarray problem
Maximum subarray problem

4 -6 7 2 5 -1 3 2

4 -6 7 2  mid  right

4  mid  9
Maximum subarray problem

\[4 \ -6 \ 7 \ 2 \ 5 \ -1 \ 3 \ 2\]

mid

right

\[4 \ -6 \ 7 \ 2\]

\[4 \ 7 \ 9\]
Maximum subarray problem

4  -6  7  2  5  -1  3  2

9  (7 2)  mid  right
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 -1 mid 3 2
Maximum subarray problem

\[ 4 \quad -6 \quad 7 \quad 2 \quad 5 \quad -1 \quad 3 \quad 2 \]

\[ 9 \quad (7 \quad 2) \quad \text{mid} \quad 5 \quad -1 \quad 3 \quad 2 \]

\[ 5 \quad -1 \quad \text{mid} \quad 3 \quad 2 \]

\[ 5 \quad \text{mid} \quad -1 \]
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 -1 mid 3 2

5 mid -1 3 mid 2
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 -1 mid 3 2

5 4 -1 3 mid 2
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 -1 mid 3 2

5 4 -1 3 5 2
Maximum subarray problem

4  -6  7  2  5  -1  3  2

9  (7 2)  mid  5  -1  3  2

5  mid  3  2

3  5  2
Maximum subarray problem
Maximum subarray problem

```
4 -6 7 2 5 -1 3 2
9 (7 2)    mid    5 -1 3 2
       |                |
       5 9 5          5 9 5
```
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2)  mid  9 (5 -1 3 2)
Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) 18 (7 2 5 -1 3 2) 9 (5 -1 3 2)
Divide and conquer solves a problem of size $n$ by:

- splitting it into $a$ subproblems of size $n/b$
- combining the answers in $O(n^d)$ time

where $a, b, d > 0$
If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}
\]
Maximum subarray problem

if high == low
    return (low, high, A[low])
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY (A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY (A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high)
if left-sum ≥ right-sum and left-sum ≥ cross-sum
    return (left-low, left-high, left-sum)
elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
    return (right-low, right-high, right-sum)
else return (cross-low, cross-high, cross-sum)
## Maximum subarray problem

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve the midpoint case</td>
<td></td>
</tr>
<tr>
<td>Solve the left and right cases</td>
<td></td>
</tr>
<tr>
<td>Find the max of those three</td>
<td></td>
</tr>
</tbody>
</table>
### Maximum subarray problem

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve the midpoint case</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Solve the left and right cases</td>
<td></td>
</tr>
<tr>
<td>Find the max of those three</td>
<td></td>
</tr>
</tbody>
</table>
## Maximum subarray problem

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve the midpoint case</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Solve the left and right cases</td>
<td>$T(n/2)$ each</td>
</tr>
<tr>
<td>Find the max of those three</td>
<td></td>
</tr>
</tbody>
</table>
### Maximum subarray problem

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve the midpoint case</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Solve the left and right cases</td>
<td>$T(n/2)$ each</td>
</tr>
<tr>
<td>Find the max of those three</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Maximum subarray problem

We get the following recurrence:

\[ T(n) = 2T(n/2) + O(n) \]

To match \( T(n) = aT(n/b) + O(n^d) \):

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)
Maximum subarray problem

If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}
\]

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)

\[
\log_b a = \log_2 2 = 1 = d
\]

Use the second case.
Maximum subarray problem

If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}
\]

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)

\[
O(n^d \log n) = O(n^1 \log n) = O(n \log n)
\]
Matrix multiplication

If $A$ and $B$ are $n \times n$ matrices, we can multiply them to produce a $n \times n$ matrix $C$:

$$A \cdot B = C$$

then $c_{ij}$ is the dot product of row $i$ of $A$ and column $j$ of $B$:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$
Matrix multiplication

Naïve method:

```python
for i from 1 to n:
    for j from 1 to n:
        c[i][j] = 0.0
    for k from 1 to n:
        c[i][j] += a[i][k] * b[k][j]
```

How long does this take?
Can we view this as a divide and conquer algorithm?

Suppose we decompose each matrix into four \(n/2 \times n/2\) blocks:

\[
A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}
\]
Then we can define multiplication as such:

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} =
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix} \cdot
\begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
\]
So we have these subproblems to compute:

\[
\begin{align*}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{align*}
\]
Matrix multiplication

\[ n = A.\text{rows} \]

let \( C \) be a new \( n \times n \) matrix

if \( n == 1 \)

\[ c_{11} = a_{11} \cdot b_{11} \]

else partition \( A, B, \) and \( C \) as in equations (4.9)

\[
\begin{align*}
C_{11} &= \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11}) \\
    &+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21}) \\
C_{12} &= \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12}) \\
    &+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22}) \\
C_{21} &= \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11}) \\
    &+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21}) \\
C_{22} &= \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12}) \\
    &+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})
\end{align*}
\]

return \( C \)
Matrix multiplication

- Divide into 8 blocks of size $\frac{n}{2} \times \frac{n}{2}$
- Recursively multiply
- Add (some of) the resulting matrices
## Matrix multiplication

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide into 8 blocks of size $n/2 \times n/2$</td>
<td></td>
</tr>
<tr>
<td>Recursively multiply</td>
<td></td>
</tr>
<tr>
<td>Add resulting matrices</td>
<td></td>
</tr>
</tbody>
</table>
Matrix multiplication

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide into 8 blocks of size $n/2 \times n/2$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recursively multiply</td>
<td></td>
</tr>
<tr>
<td>Add resulting matrices</td>
<td></td>
</tr>
</tbody>
</table>
Matrix multiplication

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide into 8 blocks of size ( n/2 \times n/2 )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Recursively multiply</td>
<td>( T(n/2) ) each</td>
</tr>
<tr>
<td>Add resulting matrices</td>
<td></td>
</tr>
</tbody>
</table>
Matrix multiplication

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide into 8 blocks of size $n/2 \times n/2$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recursively multiply</td>
<td>$T(n/2)$ each</td>
</tr>
<tr>
<td>Add resulting matrices</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
Then the time required has the form:

\[ T(n) = 8T(n/2) + O(n^2) \]
Matrix multiplication

\[ T(n) = 8T(n/2) + O(n^2) \]

Master theorem: \( T(n) = aT(n/b) + O(n^d) \)

- \( a = 8 \)
- \( b = 2 \)
- \( d = 2 \)
Matrix multiplication

If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}
\]

- \( a = 8 \)
- \( b = 2 \)
- \( d = 2 \)

\[
\log_b a = \log_2 8 = 3 > 2 = d
\]

Use case 3.
Matrix multiplication

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}$$

- $a = 8$
- $b = 2$
- $d = 2$

$$O(n^{\log_b a}) = O(n^3)$$
So matrix multiplication is $O(n^3)$.

Is $n^3$ also a lower bound?

(Is it also $\Omega(n^3)$, and thus $\Theta(n^3)$?)
So matrix multiplication is $O(n^3)$.

Is $n^3$ also a lower bound?

(Is it also $\Omega(n^3)$, and thus $\Theta(n^3)$?)

Actually, there is a clever rearrangement that can do better!

It was found in 1969 by Volker Strassen, and is known as Strassen’s method.
“Strassen’s method is not at all obvious.”

– CLRS
There are four steps to Strassen’s method:
There are four steps to Strassen’s method:

- Divide the matrices into $n/2 \times n/2$ matrices as before
Strassen’s method

There are four steps to Strassen’s method:

- Divide the matrices into $n/2 \times n/2$ matrices as before
- Add/subtract some of these to create 10 new temporary matrices $S_1, \ldots, S_{10}$
Strassen’s method

There are four steps to Strassen’s method:

- Divide the matrices into $n/2 \times n/2$ matrices as before
- Add/subtract some of these to create 10 new temporary matrices $S_1, \ldots, S_{10}$
- Compute 7 matrix products $P_1, \ldots, P_7$
Strassen’s method

There are four steps to Strassen’s method:

- Divide the matrices into \( n/2 \times n/2 \) matrices as before
- Add/subtract some of these to create 10 new temporary matrices \( S_1, \ldots, S_{10} \)
- Compute 7 matrix products \( P_1, \ldots, P_7 \)
- Compute submatrices \( C_{11}, C_{12}, C_{21}, C_{22} \)

\[
\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}
\]
Strassen’s method

First compute the $S$ matrices:

\[
\begin{align*}
S_1 &= B_{12} - B_{22}, \\
S_2 &= A_{11} + A_{12}, \\
S_3 &= A_{21} + A_{22}, \\
S_4 &= B_{21} - B_{11}, \\
S_5 &= A_{11} + A_{22}, \\
S_6 &= B_{11} + B_{22}, \\
S_7 &= A_{12} - A_{22}, \\
S_8 &= B_{21} + B_{22}, \\
S_9 &= A_{11} - A_{21}, \\
S_{10} &= B_{11} + B_{12}.
\end{align*}
\]
Then compute the $P$ matrices:

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22},$$

$$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22},$$

$$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11},$$

$$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11},$$

$$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22},$$

$$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22},$$

$$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.$$
Strassen’s method

Finally, combine these to get the $C$ submatrices:

\[ C_{11} = P_5 + P_4 - P_2 + P_6 \]
\[ C_{12} = P_1 + P_2 \]
\[ C_{21} = P_3 + P_4 \]
\[ C_{22} = P_1 + P_5 - P_3 - P_7 \]
Strassen’s method

Expanding $C_{11}$:

\[
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
- A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\
- A_{11} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{21} + A_{12} \cdot B_{22} + A_{12} \cdot B_{21} \\
\]

\[
A_{11} \cdot B_{11} + A_{12} \cdot B_{21},
\]
Expanding $C_{12}$:

$$A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$$
$$+ A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$

\[ A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \]
Expanding $C_{21}$:

$$A_{21} \cdot B_{11} + A_{22} \cdot B_{11} - A_{22} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$A_{21} \cdot B_{11} + A_{22} \cdot B_{21}.$$
Expanding $C_{22}$:

\[
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}
- A_{11} \cdot B_{22} - A_{22} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11}
- A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12}
\]

\[
A_{22} \cdot B_{22} + A_{21} \cdot B_{12}
\]
### Strassen’s method

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide the matrices</td>
<td></td>
</tr>
<tr>
<td>Add/subtract to get $S_1, \ldots, S_{10}$</td>
<td></td>
</tr>
<tr>
<td>Multiply to get $P_1, \ldots, P_7$</td>
<td></td>
</tr>
<tr>
<td>Compute $C_{11}, C_{12}, C_{21}, C_{22}$</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Time</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Divide the matrices</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Add/subtract to get $S_1, \ldots, S_{10}$</td>
<td></td>
</tr>
<tr>
<td>Multiply to get $P_1, \ldots, P_7$</td>
<td></td>
</tr>
<tr>
<td>Compute $C_{11}, C_{12}, C_{21}, C_{22}$</td>
<td></td>
</tr>
</tbody>
</table>
## Strassen’s method

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide the matrices</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Add/subtract to get $S_1, \ldots, S_{10}$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Multiply to get $P_1, \ldots, P_7$</td>
<td></td>
</tr>
<tr>
<td>Compute $C_{11}, C_{12}, C_{21}, C_{22}$</td>
<td></td>
</tr>
</tbody>
</table>
## Strassen’s method

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide the matrices</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Add/subtract to get $S_1, \ldots, S_{10}$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Multiply to get $P_1, \ldots, P_7$</td>
<td>$T(n/2)$ each</td>
</tr>
<tr>
<td>Compute $C_{11}, C_{12}, C_{21}, C_{22}$</td>
<td></td>
</tr>
</tbody>
</table>
Strassen’s method

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide the matrices</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Add/subtract to get $S_1, \ldots, S_{10}$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Multiply to get $P_1, \ldots, P_7$</td>
<td>$T(n/2)$ each</td>
</tr>
<tr>
<td>Compute $C_{11}, C_{12}, C_{21}, C_{22}$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
So the overall time is

\[ T(n) = 7T(n/2) + O(n^2) \]
Strassen’s method

\[ T(n) = 7T(n/2) + O(n^2) \]

- \( a = 7 \)
- \( b = 2 \)
- \( d = 2 \)

\( \log_2 7 \approx 2.807 > d \)
Strassen’s method

\[ T(n) = 7T(n/2) + O(n^2) \]

- \( a = 7 \)
- \( b = 2 \)
- \( d = 2 \)

\[ \log_2 7 \approx 2.807 > d \]

So using case 3 of master theorem, \( T(n) = O(n^{\log_2 7}) \approx O(n^{2.807}) \)
So is matrix multiplication $\Theta(n^{2.807})$?
So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.
Matrix multiplication

So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
- $O(n^{2.376})$ algorithm found in 1990 (Coopersmith-Winograd)
Matrix multiplication

So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
- $O(n^{2.376})$ algorithm found in 1990 (Coopersmith-Winograd)
- $O(n^{2.3729})$ in 2013 (Virginia Williams)
Matrix multiplication

So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
- $O(n^{2.376})$ algorithm found in 1990 (Coopersmith-Winograd)
- $O(n^{2.3729})$ in 2013 (Virginia Williams)
- $O(n^{2.3728639})$ in 2014 (François Le Gall)
Matrix multiplication

So is matrix multiplication $\Theta(n^{2.807})$?

Let $\omega$ denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
- $O(n^{2.376})$ algorithm found in 1990 (Coopersmith-Winograd)
- $O(n^{2.3729})$ in 2013 (Virginia Williams)
- $O(n^{2.3728639})$ in 2014 (François Le Gall)

$2 \leq \omega < 2.373$, but we still don’t know if $\omega = 2$ is possible!
Another clever rearrangement

Suppose we have two complex numbers, $a + bi$, and $c + di$. We can multiply them as such:

$$(a + bi)(c + di) = ac + adi + bci + bdi^2$$

$$= ac + (ad + bc)i - bd$$

$$= ac - bd + (ad + bc)i$$
Another clever rearrangement

\[(a + bi)(c + di) = ac - bd + (ad + bc)i\]

So we have to do four multiplications.
Another clever rearrangement

\[(a + bi)(c + di) = ac - bd + (ad + bc)i\]

But Gauss noticed that

\[ad + bc = (a + b)(c + d) - ac - bd\]
Another clever rearrangement

\[(a + bi)(c + di) = ac - bd + (ad + bc)i\]

\[ad + bc = (a + b)(c + d) - ac - bd\]

So we can compute this with only three multiplications!
Another clever rearrangement

Both are $\Theta(1)$, so does this matter?
Both are $\Theta(1)$, so does this matter?

Let’s consider multiplying $n$-bit binary numbers...
Let's consider multiplying $n$-bit binary numbers $x$ and $y$. And let's assume $n$ is a power of 2, for simplicity. Can we divide and conquer this problem?
Multiplying $n$-bit numbers

Divide: split $x$ and $y$ into two pieces of length $n/2$

\[
x = x_L \quad x_R = 2^{n/2} x_L + x_R
\]

\[
y = y_L \quad y_R = 2^{n/2} y_L + y_R.
\]

E.g., if $x = 10110110_2$, this is also equal to $2^4 \cdot 1011_2 + 0110_2$.  

62
Multiplying $n$-bit numbers

Divide: split $x$ and $y$ into two pieces of length $n/2$

$$(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2} x_R y_L + 2^{n/2} x_L y_R + x_R y_R$$

$$= 2^n x_L y_L + 2^{n/2} (x_R y_L + x_L y_R) + x_R y_R$$

We need four recursive multiplications.
Multiplying $n$-bit numbers

This gives a running time of

$$T(n) = 4T(n/2) + O(n)$$

which is $O(n^2)$ by the master method.
Let’s try Gauss’s trick:

\[(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^nx_Ly_L + 2^{n/2}(x_Ry_L + x_Ly_R) + x_Ry_R\]

\[x_Ry_L + x_Ly_R = (x_L + x_R)(y_L + y_R) - x_Ly_L - x_Ry_R\]
Multiplying \( n \)-bit numbers

Now with only three recursive multiplications, this gives a running time of

\[
T(n) = 3T(n/2) + O(n)
\]

which is \( O(n^{\log_2 3}) \approx O(n^{1.59}) \).
The goal is to move all discs from peg A to C, but a larger disc can never go on top of a smaller one.

Image credit: Evanherk

The idea behind the recursive solution:

● If we could somehow get all $n - 1$ discs from A to B, then we could move the largest disc to C.
● Then we could move all $n - 1$ discs from B to C.
moveDiscs(src, dest, n):
  if n == 1:
    move top disc from src to dest
  otherwise:
    moveDiscs(src, otherPeg, n – 1)
    move remaining disc from src to dest
    moveDiscs(otherPeg, dest, n – 1)
## Tower of Hanoi

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move from src to other</td>
<td></td>
</tr>
<tr>
<td>Move one disc</td>
<td></td>
</tr>
<tr>
<td>Move from other to dest</td>
<td></td>
</tr>
</tbody>
</table>
## Tower of Hanoi

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move from src to other</td>
<td>$T(n - 1)$</td>
</tr>
<tr>
<td>Move one disc</td>
<td></td>
</tr>
<tr>
<td>Move from other to dest</td>
<td></td>
</tr>
</tbody>
</table>
## Tower of Hanoi

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move from src to other</td>
<td>$T(n - 1)$</td>
</tr>
<tr>
<td>Move one disc</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Move from other to dest</td>
<td></td>
</tr>
</tbody>
</table>
## Tower of Hanoi

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move from src to other</td>
<td>$T(n-1)$</td>
</tr>
<tr>
<td>Move one disc</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Move from other to dest</td>
<td>$T(n-1)$</td>
</tr>
</tbody>
</table>
So we get a time of

\[ T(n) = 2T(n - 1) + O(1) \]
So we get a time of

\[ T(n) = 2T(n - 1) + O(1) \]

But it’s not in the form of the master theorem. Now what?
We can solve this using another method, called recursion trees.

\[ T(n) \]
We can solve this using another method, called recursion trees.

\[
\begin{array}{c}
c \\
| \downarrow \quad \downarrow |
\end{array}
\begin{array}{cc}
T(n - 1) & T(n - 1)
\end{array}
\]
Expanding one level:

Continue until we reach the leaves.
How many levels are there?

How much work is done at each level?

What is the sum of all levels?

75
Tower of Hanoi

- How many levels are there?

- How much work is done at each level?

- What is the sum of all levels?
Tower of Hanoi

- How many levels are there?
- How much work is done at each level?
• How many levels are there?
• How much work is done at each level?
• What is the sum of all levels?

75
• How much work is done at each level?

• What is the sum of all levels?
Tower of Hanoi

• How much work is done at each level?
  • $2^k$ nodes, each with $c$ work, so $c2^k$

• What is the sum of all levels?
Tower of Hanoi

- How much work is done at each level?
  - $2^k$ nodes, each with $c$ work, so $c2^k$

- What is the sum of all levels?

$$\sum_{k=0}^{n-1} c2^k = c(2^n - 1)$$
• How much work is done at each level?
  • $2^k$ nodes, each with $c$ work, so $c2^k$

• What is the sum of all levels?

$$\sum_{k=0}^{n-1} c2^k = c(2^n - 1)$$

This implies that the recursive algorithm takes $O(2^n)$ steps.
Tower of Hanoi

Another way to see this:

<table>
<thead>
<tr>
<th>$n$</th>
<th>#steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
</tbody>
</table>
We can guess that this is $2^n - 1 = O(2^n)$, but we have to prove it.
We can guess that this is $2^n - 1 = O(2^n)$, but we have to prove it. Use induction!
Claim: The recursive algorithm takes $2^n - 1$ steps.
Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
- Inductive case:
Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
- Inductive case:
  - Assume $T(n - 1) = 2^{n-1} - 1$
Claim: The recursive algorithm takes $2^n - 1$ steps.

- **Base case:** $n = 1$, easy to check.
- **Inductive case:**
  - Assume $T(n - 1) = 2^{n-1} - 1$
  - Use the inductive hypothesis to prove $T(n) = 2^n - 1$:

\[
T(n) = 2T(n - 1) + 1
\]

  \[
  = \\
  = \\
  =
\]
Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
- Inductive case:
  - Assume $T(n-1) = 2^{n-1} - 1$
  - Use the inductive hypothesis to prove $T(n) = 2^n - 1$:

$$T(n) = 2T(n-1) + 1$$
$$= 2(2^{n-1} - 1) + 1$$
$$= 2^n - 1$$
Claim: The recursive algorithm takes \(2^n - 1\) steps.

- Base case: \(n = 1\), easy to check.
- Inductive case:
  - Assume \(T(n - 1) = 2^{n-1} - 1\)
  - Use the inductive hypothesis to prove \(T(n) = 2^n - 1\):

\[
T(n) = 2T(n - 1) + 1 \\
  = 2(2^{n-1} - 1) + 1 \\
  = 2^n - 2 + 1 \\
  = \;
\]
Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
- Inductive case:
  - Assume $T(n - 1) = 2^{n-1} - 1$
  - Use the inductive hypothesis to prove $T(n) = 2^n - 1$:

$$T(n) = 2T(n - 1) + 1$$
$$= 2(2^{n-1} - 1) + 1$$
$$= 2^n - 2 + 1$$
$$= 2^n - 1$$
Summary

- Divide and conquer is a powerful tool for designing algorithms
- Often get a recurrence of the form \( T(n) = aT(n/b) + O(n^d) \)
- If so, use the master theorem
- If not, draw the recursion tree and sum the work at each node
- Sometimes you’ll get a general answer with some coefficients
- Use induction to solve and verify